A GENERAL SIMULATION MODEL FOR INFORMATION SYSTEMS: A REPORT ON A MODELLING CONCEPT

A. L. Buchanan and R. B. Waina

July 1969

P-4140

18

A GENERAL SIMULATION MODEL FOR INFORMATION SYSTEMS: A REPORT ON A MODELLING CONCEPT

A. L. Buchanan and R. B. Waina*

The Rand Corporation, Santa Monica, California

We have been concerned for some time with the design of management information systems (MIS) in general and lately in particular with a class of MIS that we call large-scale. Recent involvement in and observation of design efforts by the Air Force Logistics Command motivated us to create a special design methodology to use in these situations, and a design model to complement it. The purpose of this paper is to present the model; therefore, we will describe the approach only in outline form with no justification. But before we do that, a definition of large-scale seems in order.

DESIGN APPROACH

There are a number of definitions of large-scale systems, but the one we favor, because it fits the design problem so well, is as follows: a large-scale system is one that no one person or small number of people (say three or four) can understand in totality in any operational sense (e.g., know it well enough to be able to design a MIS to support it.) The implication of this definition for a design effort is a very large

*Any views expressed in this paper are those of the authors. They should not be interpreted as reflecting the views of The Rand Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The Rand Corporation as a courtesy to members of its staff.

This paper is being prepared for presentation at the Third Conference on the Applications of Simulation, sponsored by ACM, AIIE, IEE, SCI and TIMS, in Los Angeles, December 1969.

design team requiring formal design and control aids. From experience we have devised an approach to design that we will briefly outline in step form.

1. A small number of multiskilled people are formed into a central design team. They begin to form alternative designs at a very high policy level and, where required, they bring in other specialists to provide specific technical information.

2. Alternative policies are combined and evaluated in an iterative manner until some insight is gained about what is good policy and what is bad, and which policies fit together and which do not.

3. The iterative process continues, but now there is more interest in adding detail to the system description than comparing policies. It may very well be, though, that additional detail will cause the designers to reevaluate some policy that they had previously decided to adopt.

4. This process ends when the amount of detail becomes too great for the central group to handle. The central design group now becomes the central design control group. A description of the design is prepared and segmented into pieces. Various subdesign groups are assigned to each piece to work under the direction of the central design control group to finish the design by completing the detail.

At the transition point from one design group to many, the design is well established and only very serious nondesign occurrences cause it to change in a basic way. Before the transition is made, the design effort is one that is well understood and documented. It is the transition, segmenting and control problems that are little understood and that require formal aids. This paper proposes a simulation model as an approach to all these problems.

In summary, we subscribe to the philosophy that you start with some overall design, break it into subsystems, and examire each subsystem in detail, including its effect on other subsystems and the total system. Further, some central control group has overall responsibility for the integrity of the final design.

Modelling Concepts

Out of two streams, then--the specific design problem and a general research interest--comes a first primitive attempt to gain insight by actually building a model. The problem we addressed is this: Develop a general model for evaluating the responsiveness of management information systems. The model would consist of a small number of primitives that an analyst could use to describe a large number of systems. While this requires a highly skilled user, it does allow sub-designs to be developed in varying levels of detail and, hopefully, connected in any of various combinations. This is necessary to be able to fit any useful model into current computers.

The model developed from such an effort could thus serve as a useful tool to system designers in the Air Force, and hopefully to MIS designers in almost any context, as our concept was to develop a system representation that would be highly flexible and context-free.

With regard to the model, we can ask three significant questions. We attempt to answer the first in this paper.

1. Is it possible to build a flexible, context-free evaluation model applicable to a variety of systems?

2. How large a set of systems can be practically handled by a single model?

3. Can such a flexible, context-free model be made sufficiently easy to use that its employment will significantly increase the quality of a system design for a given level of effort?

We assert that the answer to the first question is yes. We have achieved flexibility and context-freeness by focussing on modelling the information flow.

There are a number of techniques that involve flows in networks: industrial dynamics and DYNAMO, GERT, signal flowgraphs, to name a few. These approaches deal with aggregations, however, and require the development of some set of equations or some aggregate description of information flows. And such quantitative descriptions of information flows are often very difficult to come up with, and frequently require heroic assumptions by the system analyst. We therefore decided to deal in terms of individual messages, which can have a one-to-one correspondence with the real world.

Our model thus consists basically of management transforms with various kinds of messages flowing among and between them. A transform is a black box with a set of input messages, a set of output messages, and some relationship between them. In our current primitive model this relationship is merely a time delay; after all expected input messages are in, all specified output messages are created. Each individual message has its associated time delay, which we call processing time. In addition, each message has a transmission time, the time to get from its originating transform to its destination transform. Both times can be random variables from any of several specified continuous distributions.

Transforms can be grouped into nodes. This is usually done when there is some set of resource constraints, causing transforms to vie for resources. A node might be considered, for example, as a particular decisionmaker, and the transforms as the various kinds of decisions he makes. The input messages would be the information and requirements that stimulate decisions, and the output messages would be the commands, reports, and so on issuing as results of the decisions. Although we speak of messages throughout this paper, it should be realized that a message can be a proxy for any of a large class of temporary entities, such as an order, a customer, a job, or an airplane flight. By thus adopting an appropriate frame of reference, one can apply the model to a fairly large set of problems.

When using our model, the analyst describes the system under consideration using the form shown in Fig. 1. One form corresponds to one transform at a node. Any output message described on a form appears on some other form as an input message. The system relationships are thus described in terms of how transforms are tied together by messages.

In Fig. 1, "Node" is the identification number of a particular node. Nodes are numbered arbitrarily from 1 up through the number of nodes; i.e., if there were ten nodes in a system they would be numbered 1, 2, 3,..., 10.

"Transform" is any identifying symbol meaningful to the analyst. It is not used by the model.

"Type" is an arbitrary number specifying the type of information a message carries. Numbers 900 and above are reserved for "artificial" messages that accomplish various special tasks. Such messages do not enter into the simulation statistics. Substantive messages describing the system being simulated are hus limited to numbers 1 through 899.

MANAGEMENT INFORMATION SYSTEM

DESCRIPTION DATA

NODE _____ TRANSFORM _____

INPUT MESSAGES:

Type	Level	Origin	Priority	Quality

OUTPUT MESSAGES:

						Process Time			Transmit Time		
Type	Level	Dest.	Pri.	Pct.	Rsc.	Type	Mean	S.D.	Type	Mean	S. D.

Fig. 1

"Level" is an arbitrary number saying something about the information content of messages within a given type. For example, if type 85 is a message relating to inventory, level 1 might indicate that the reorder point has been reached, whereas level 2 would indicate that no order action need be taken.

"Origin" is the number of the node from which a particular input message is expected. At any given node, only one message of a particular type, level, and origin may be expected. Duplicate messages are not permitted, even though they may be in different transforms. The presence of a duplicate message will result in an error printout and suppression of the run execution.

"Destination" is the number of the node to which a particular output message is being sent. More than one message of a particular type, level and destination may be sent from a node. Duplication will result in an error message, but run execution will not be suppressed.

"Priority" designates the urgency of a message. The model logic does not yet include consideration of priority. It could be easily implemented by defining the message queues as ranked.

"Quality" is considered in the Shannon (information theory) sense; i.e., a low quality message would have high noise and be difficult to read. A low quality message might require more processing resources or more time to process, or might require retransmission of the message. Quality has a range of 0 to 1. The model logic does not yet include consideration of quality.

"Pct" is used when the node is of an or-output rather than an and-output type. When implemented, it will be the percentage of times any one message is sent out. The particular message outputted would be determined randomly.

"Rsc" is the amount of resources required to process (generate) a particular output message.

"Process Time-Type" is an integer denoting the particular distribution of the process time. The available distributions are

0 - constant

1 - normal

2 - lognormal

3 - exponential

4 - negative exponential

5 - poisson

6 - geometric

7 - weibull

"Process Time - Mean" is the mean of the distribution. "Process Time - Std Dev." is the standard deviation.

"Transmit Time" is the length of time a communication channel is occupied in transmitting a message. If messages are being transmitted by electrical means (i.e., speed of light), this time is then the length of a particular message multiplied by an appropriate conversion factor. If some slower means of communication is being used, this is the length of time it takes to physically transport the message (or other entity) from its origin to its destination.

"Transmit Time - Type" uses the same set of codes as process time type. "Transmit Time - Mean" and "Transmit Time - Std. Dev." have the obvious meanings.

It is sometimes necessary to describe a transform that periodically issues a message (or set of messages) without any input from any other transform. In this case an "artificial" message of type 900 or greater is used. The input to the transform is the artificial message, whose origin is the node the transform belongs to. The output from the transform is the set of messages plus the artificial message whose destination is the same node. The transmit time of the artificial message is the length of the period. This time can be either a constant or a random variable.

In its current stage of development, the model can be used to describe two general types of networks when channel constraints exist (see Fig. 2):

1. Every node can communicate with every other node through a central switching node. The constraints are

   a. The number of input channels at each node.

   b. The number of output channels at each node.

   c. The number of duplex channels at the central node.

This central node is not explicitly modeled, but is treated implicitly within the program logic.

2. Nodes are arranged in a single string, so that each node has only one predecessor node and one successor node. The constraints are items a and b above. The number of output channels at each node must equal the number of inputs at its successor.

When there are no channel constraints, virtually any type of network can be modeled (see Fig. 2b).

The model is written in SIMSCRIPT I.5 on an IBM 360/65. We chose that language primarily because of our familiarity with it, and also because it seems to be a good match with our modelling concepts.

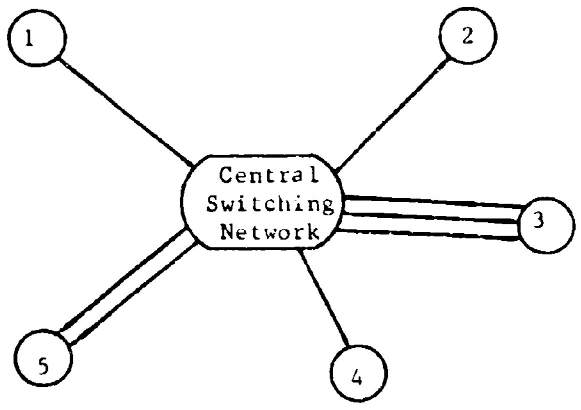


Fig. 2

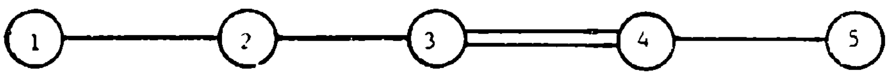


Fig. 2a

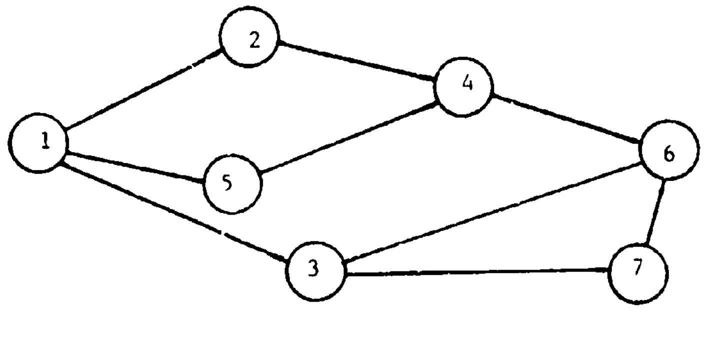
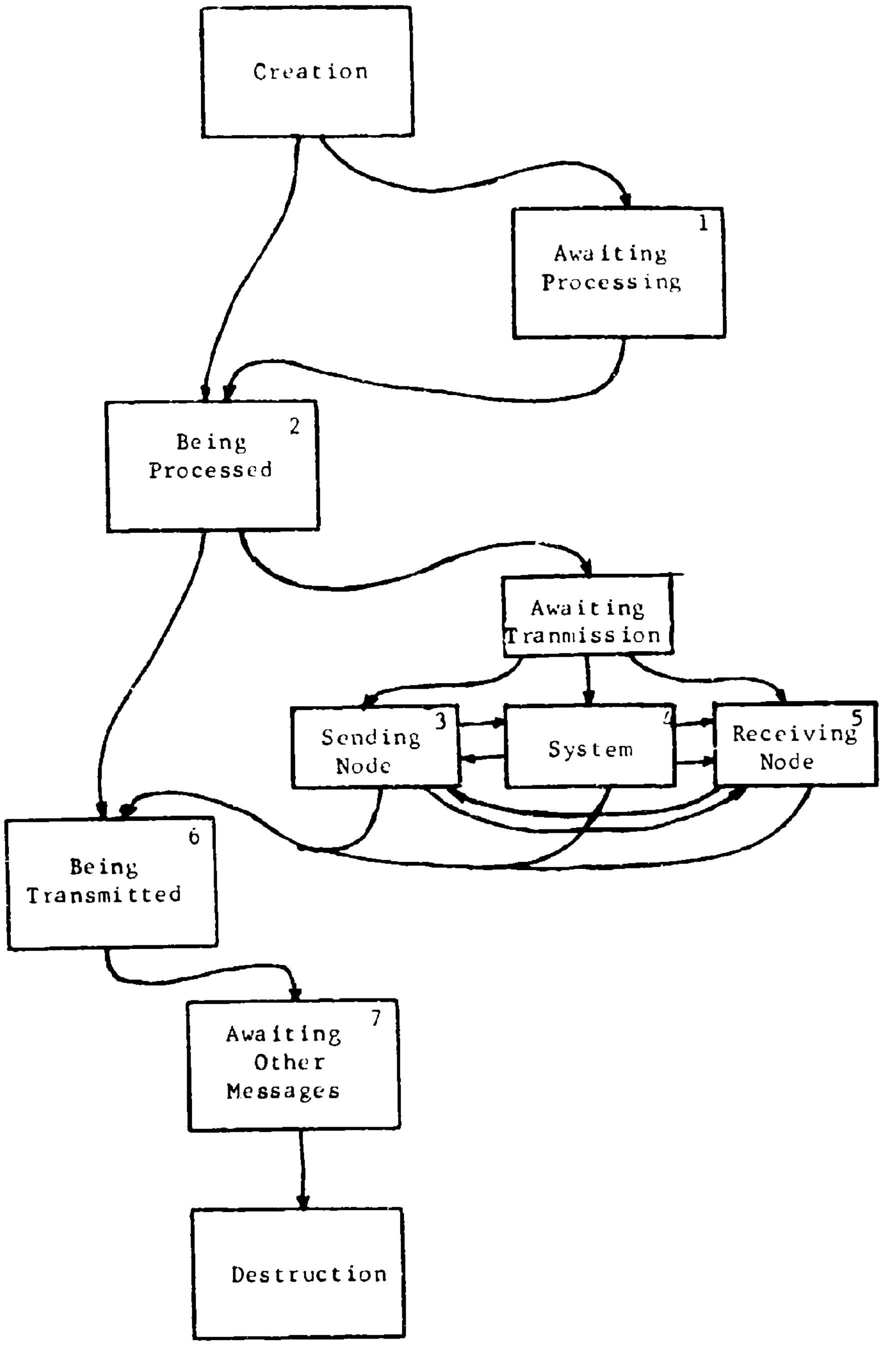


Fig. 2b

The cards punched from the forms (Fig. 1) describing the system are read by routine START. This routine, after establishing the system and associated parameters, examines the system for missing and duplicate input and output messages and indicates any such errors. If possible, it then starts the system in operation by creating one of each possible output message. The network thus starts out in a relatively full state and seeks its steady state from there. It is also possible to start the system in a relatively empty state and let it build up. Care must be taken to insure that the steady state can be reached from the starting state (e.g., periodic messages must be generated initially or they will never be generated).

The prime mover of this simulation model is the message as it flows through and between transforms. A message may be in any one of seven states at any given time. A diagram of the various states and their relationships is shown in Fig. 3.

A message is created. At this point it wants to begin using some processing resources. If said resources are available, it assumes state state 2 immediately. If not, it assumes state 1, and is filed in a queue of messages awaiting processing resources at that particular node. Upon completion of processing the message is ready to be transmitted to its destination. If a channel is free, the message assumes state 6. If not, it is filed in one of several queues of messages awaiting transmission. If there are no output channels available at the node of origin, the message assumes state 3 and is filed in a queue at the origin. If there are no channels available at the central switching node (assuming that type of system is being modelled), the message assumes state 4 and is filed in a queue at the destination. If there are no input channels



The Life of a Message

Fig. 3

available at the node of destination, the message assumes state 5 and is filed in a different queue at the destination. A message can experience several shifts between states 3, 4, and 5 before it finally assumes state 6. Upon completion of transmission, the message assumes state 7. It remains in this state until all expected input messages have arrived at the transform. These input messages are then destroyed, and the related set of output messages created.

As an example of the logic at a transform, assume that a particular transform has three input messages and two output messages. Associated with each transform is a counter. In this example the counter would initially be set equal to three. When a message arrives and is identified as being associated with this particular transform, the transform's counter is decremented by one. When the counter is equal to zero, this indicates the arrival of all three messages. They would then be destroyed, and the output messages of that transform created. The counter would be reset to three to await another set of input messages.

The counting logic is such that duplicate messages are ignored until the counter is reset. For example, assume a particular transform has two inputs, one arriving at (simulated) weekly intervals and one at monthly intervals. When the first weekly message arrives, the counter is decremented from two to one. When the second, third, and fourth weekly messages arrive, the counter is not decremented. When the monthly message arrives, the counter is decremented from one to zero, the input messages destroyed, and the output messages created.

The transmission logic uses pairs of counters. One counter contains the number of channels available at a node or in the system. The second counter in the pair contains the number of channels currently in use. When the two are equal a request for the use of that particular set of channels is denied. For a message to be transmitted, a channel must be available from each of three sets of channels: the output channels at the origin; the trunk channels at the central switching node; the input channels at the destination. Channels are queried in that order.

The purpose of a simulation is to study the behavior of the system under varying conditions. To do this it is necessary to collect statistics on various aspects of system operation. In this model the following statistics are collected by node:

By message state, for all messages:
- total number during run
- maximum number in queue at any one time
- average number in queue
- maximum time any message is in queue
- average time messages are in queue

Processing resource utilization:
- maximum amount of resources used
- average amount of resources used
- percentage of time all resources are idle

Communication channel utilization:
- maximum number of channels used
- average number of channels used

Time spent waiting for communication channels:
- average time spent
- number of messages flowing between nodes

Another statistic collected is the flow time of a sequence (or chain) of messages. (A sequence may consist of one or more messages.) Flow time is the average length of simulated time from the creation of the first message in a sequence until the arrival at its destination of the last message in a sequence. It does not include the time the last message (only) is in state 7.

Application of the Model

To demonstrate its versatility the model described here has been applied to three different systems: a management system, a communication system, and a job shop. In each, the primary modeling difficulty was data collection.

The management system chosen was the reparable parts portion of a proposal for the Air Force Logistics Command (AFLC). The particular abstraction of that system we attempted to model consisted of eleven different planning or operating cycles, with eighteen different kinds of data flowing among them. Conceptually, the model was quite simple. However, obtaining numbers to describe the transmission and processing times proved difficult. This difficulty led to the next method of application of the model, which is to describe each transform in more detail, in terms of the messages and transforms within a transform. That is, we describe a subsystem in terms of *its* subsystems and relationsnips, and run this model of the subsystem. Based on this analysis, we can then obtain empirical distributions of the relation between inputs and outputs, which can be used as describing numbers in the higher level model. This more detailed analysis is now being carried out.

The second system modeled was the communication system between the depots and the headquarters of the AFLC. This was done based on a projection of traffic in the system during the 1970s. In this application, however, data were not readily available on the relationships between messages, so we had to be content with looking at traffic flow and queue buildup at transmission channels. The system consisted of the headquarters, five major depots, and two minor depots, yielding eight nodes. Associated with these were over 300 individual transforms, and almost 800 messages flowing at daily, weekly, monthly, quarterly, semi-annual and annual intervals. The actual description of the system on the forms was done by an eighteen-year-old research assistant in less than two days. Thus, even though a model written specifically for that system might have been more computationally efficient, its development and debug expense would probably have been considerably greater. Running time of this system on the 360/65 for a simulation period of two years was 400-600 seconds, and the core requirement was 228 K bytes.

The third model was an aircraft engine repair shop with three different kinds of engines going through four repair processes. Arrivals at the shop as well as repair times can be from any specified distribution. Again, system description was very rapid and easy.

CONCLUSIONS

We have thus demonstrated that it is possible to build a flexible, context-free evaluation model applicable to a variety of systems. This was done by creating a model that represents information flows, and then describing any particular system in terms of its information flows. The model collects various standard queueing statistics to describe the performance of any particular system. It also measures the system's response performance in terms of the total flow time of sequences of messages. The model can thus be used to study various system designs in order to gain insight into their relative performance. It should therefore be a useful tool to the system designer.

The utility of the model could be enhanced by several refinements. The major one is the inclusion of a decision table in the transform description. This addition would make possible a much better description of a decision transform, and also simplify the analyst's task by making the description correspond more closely to reality. Allied with this change would be the addition of a representation of a generalized data base whose values could be changed by messages. The data base could itself serve as one of the inputs to the decision table. In addition, the data base could serve as a means of dynamically changing system parameters (such as resource availability) during a run. Finally, some sort of data collection routines must be developed that can be modified by the system analyst to suit his particular application. Such a set of capabilities should greatly enlarge the set of systems that the model described and thus make it a powerful tool for the information system designer.